From Micro-Analytical Method to Mass Processing - The Economic Challenge

Henrard, Jean; Hainaut, Jean-Luc; Hick, Jean-Marc; Roland, Didier; Englebert, Vincent

Published in:
Proc. of Data Reverse Engineering Workshop 2000 (DRE'2000)

Publication date:
2000

From micro-analytical Techniques to Mass Processing in Data Reverse Engineering The Economic Challenge

J. Henrard, J.-L. Hainaut, J.-M. Hick, D. Roland, V. Englebert
Institut d'Informatique, University of Namur
rue Grandgagnage, 21 - B-5000 Namur - Belgium
tel: +32 81 724985 - fax: +32 81 724967
db-main@info.fundp.ac.be

Abstract

Database reverse engineering (DBRE) attempts to recover the technical and semantic specifications of the persistent data of information systems. Despite the power of the supporting tools, the discovery of a single foreign key through the analysis of half a million lines of code may require several hours of meticulous analysis. Considering that an actual schema can include several thousands of implicit constructs, DBRE can prove very costly for medium to large projects.

Even in the parts of the process that can be automated, tools must be used with much caution. While they can help the analyst, the latter needs to understand their strength and weakness.

On the other hand, reducing the quality requirements of the result can lead to much higher costs when the recovered conceptual schema is used as the basis for reengineering the information system or to migrate its data to datawarehouses or to other applications.

Hence the dilemma: how can we automate the highly knowledge-based and interactive DBRE process without impairing the quality of the resulting products?

1. Introduction

Reverse engineering a piece of software consists, among others, in recovering or reconstructing its functional and technical specifications, starting mainly from the source text of the programs. Recovering these specifications is generally intended to redocument, convert, restructure, maintain or extend legacy applications.

In information systems, or data-oriented applications, i.e., in applications the central component of which is a database (or a set of permanent files), it is generally considered that the complexity can be broken down by considering that the files or database can be reverse engineered (almost) independently of the procedural parts, through a process called *DataBase Reverse Engineering* (DBRE in short).

This proposition to split the problem in this way can be supported by the following arguments.

- The semantic distance between the so-called conceptual specifications and the physical implementation is most often narrower for data than for procedural parts.
- The permanent data structures are generally the most stable part of applications.
- Even in very old applications, the semantic structures that underlie the file structures are mainly procedure-independent (though their physical structures may be highly procedure-dependent).
- Reverse engineering the procedural part of an application is much easier when the semantic structure of the data has been elicited.

Therefore, concentrating on reverse engineering the application data components first can be much more efficient than trying to cope with the whole application.

Even if reverse engineering the data structure is *easier* than recovering the specification of the application as a whole, it is still a complex and long task. The elicitation of implicit constructs (including hidden constraints) is both complex and tedious. As an illustration of this complexity, about fifteen essential constructs have been identified, and more than ten major techniques have been proposed to elicit them.

Currently DBRE problems are fairly well understood and techniques are available to solve them. However, most tools supporting these techniques are *punctual*, i.e., they address one instance of a data structure pattern to be recovered, so that the analyst has to apply them iteratively. For example, when looking for foreign keys, (s)he has to apply the *foreign key assistant* on each identifier to find the possible foreign keys that reference this identifier. These tools and techniques can easily be applied to small projects (say, 50,000 LOC and 20 files). But they are unusable for larger projects that can have several millions LOC and several hundreds files or tables.

We can easily be convinced that the complexity of DBRE projects is between $O(V)$ and $O(V^3)$, where V represents some measure of the size of the legacy system. For

instance, each file/table can be semantically related with any of the tables of the schema. Therefore, we have to examine whether each couple of tables is linked through one or several foreign keys. In addition, each potential foreign key requires the examination of the source code of the programs. We can make the hypothesis that the huger the application, the larger the database. This leads to a process with complexity $O(V^3)$, where V is the size of the application, i.e. some measurement of the number of files/tables in the database and the number of LOC. Fortunately, some structures require a more lower complexity, so that we can state that, considered as a whole, the data reverse engineering process has a complexity $O(V^2)$.

The bad news is that the cost of a DBRE project, also is a function of the square of V, which lead to an unacceptable high cost for large projects. One way to reduce this cost is to automate the process. But as shown latter, this automation cannot be complete. Even in parts of the process that can be automated, tools must be used with much caution and with the analyst supervision.

This paper is organized as follows. Section 2 is a synthesis of a generic DBMS-independent DBRE methodology. Section 3 explains how to decide that a DBRE project is finished. Section 4 and 5 present the need for automation and its limits. In section 6, DBRE project evaluation is discussed. The economic challenge is presented in section 7. Section 8 shows the evolution of the DBRE approach in DB-MAIN project and CASE tool.

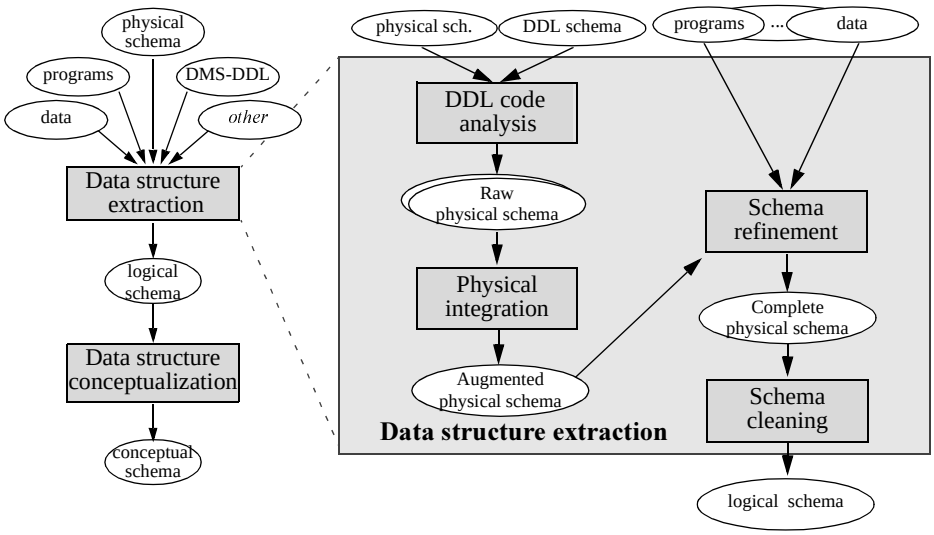

Figure 1. The major processes of the reference DBRE methodology (left) and the development of the data structure extraction process (right).

2. A Generic Methodology for Database Reverse Engineering

The reference DBRE methodology [3] is divided into two major processes, namely *data structure extraction* and *data structure conceptualization* (Figure 1, left). These problems address the recovery of two different schemas and require different concepts, reasoning and tools. In addition, they grossly appear as the reverse of the physical and logical design usually considered in database design methodologies [1].

2.1. Data Structure Extraction

The first process consists in recovering the complete DMS schema, including all the explicit and implicit structures and constraints, called the *logical schema*.

It is interesting to note that this schema is the document the programmer must consult to fully understand all the properties of the data structures (s)he intends to work on. In some cases, merely recovering this schema is the main objective of the programmer, who will find the conceptual schema itself useless.

In this reference methodology, the main processes of data structure extraction are the following (Figure 1, right):

- *DDL code analysis*: parsing the data structure declaration statements to extract the explicit constructs and constraints, thus providing a *raw physical schema.*
- *Physical integration*: when more than one DDL source has been processed, several extracted schemas can be available. All these schemas are integrated into one global schema. The resulting schema (*augmented physical schema*) must include the specifications of all these partial views.

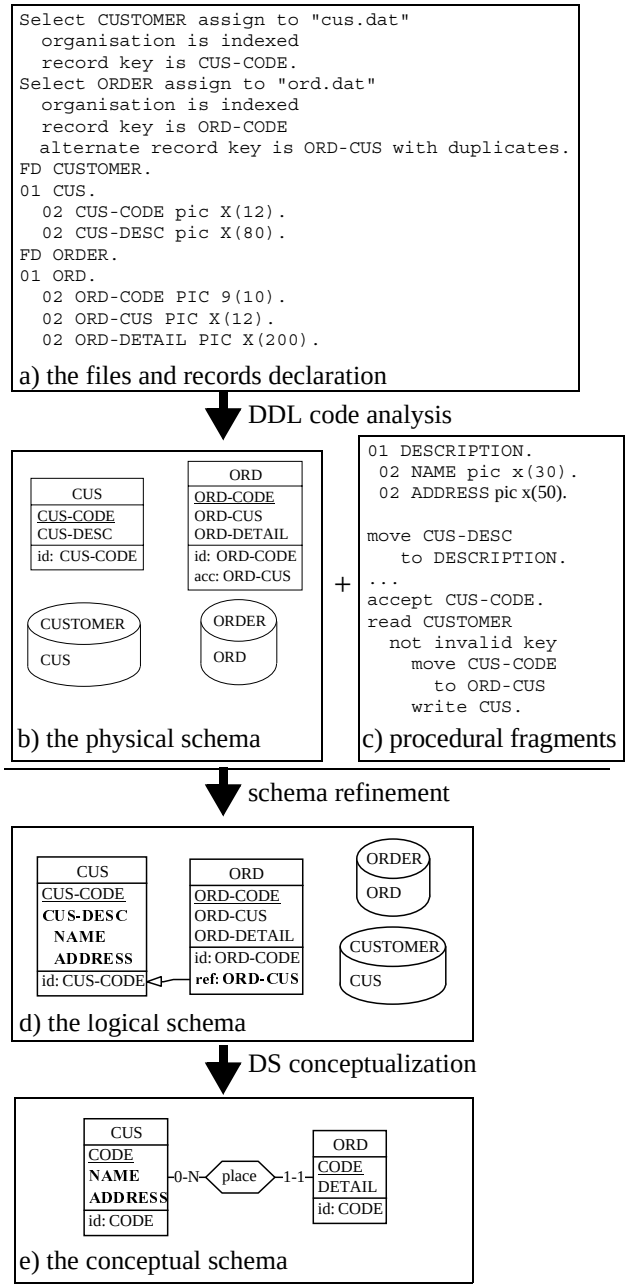

Figure 2. Database reverse engineering example.

- *Schema refinement*: the main problem of the data structure extraction phase is to discover and to make explicit the structures and constraints that were either implicitly implemented or merely discarded during the development process. The physical schema is enriched with implicit constructs made explicit, thus providing the *complete physical schema.*
- *Schema cleaning*: once all the implicit constructs have been elicited, technical constructs such as indexes or clusters are no longer needed and can be discarded in order to get the *complete logical schema* (or simply the logical schema).

The final product of this phase is the complete logical schema, which includes both explicit and implicit structures and constraints. This schema is no longer DMS-compliant for at least two reasons. Firstly, it is the result of the integration of different physical schemas, which can belong to different DMS. For example, some part of the data can be stored into a relational database, while others are stored into standard files. Secondly, the complete logical schema is the result of the refinement process that enhances the schema with recovered implicit constraints, which are not necessarily DMS compliant.

2.2. Data Structure Conceptualization

The second phase addresses the conceptual interpretation of the logical schema. It consists for instance in detecting and transforming or discarding non-conceptual structures, redundancies, technical optimization and DMS-dependent constructs.

The final product of this phase is the conceptual schema of the persistent data of the application. More detail can be found in [4].

2.3. An Example

Figure 2 gives a DBRE process example. The files and records declarations (2.a) are analyzed to yield the raw physical schema (2.b). This schema is refined through the analysis of the procedural parts of the program (2.c) to produce the logical schema (2.d).

This schema exhibits two new constructs, namely the refinement of the field CUS-DESC and a foreign key. It is then transformed into the conceptual schema (2.e).

3. How to decide that data structure extraction is completed

The goal of the reverse engineering process has a great influence on the output of the data structure extraction process and on its ending condition.

The simplest DBRE project can be to recover only the list of all the records with their fields. All the other constraints, such as identifiers, field domains and foreign keys are ignored. This can be useful to make a first inventory of the data used by an application in order to prepare another reverse or maintenance project (as Y2K conversion). In this kind of project, the partial logical schema is the final product of DBRE.

On the other hand, we can try to recover all the possible constraints to have a complete and precise schema of the

database. This schema contains records and fields decomposition, the exact domains of the fields, foreign keys, dependencies among fields, etc. For example, this can be necessary in a reengineering project where we want to convert a collection of standard files into a relational database.

Forward engineering process is finished when all the required functions of the system are correctly implemented. In DBRE, we don't have such criteria. We don't have a reference schema with which the logical schema produced by the data structure extraction process could be compared and evaluated. We cannot prove that we have elicited all the implicit constructs, since a more in-depth analysis could lead to the discovery of a new one.

But we have to finish the process, so we need some ending criteria. A possible criterion could be to decide in advance which kind of constraints we are looking for, which techniques/tools we will use to discover them and which sources of information we will analyze. When we have applied all the techniques/tools on all the information sources, then we decide that the process is finished.

Of course this criterion does not guarantee that all the constraints have been found. Moreover, determining which kind of techniques/tools to apply, to elicit, in a given project, all the constraints (s)he is looking for, relies on the analyst.

In another approach, the analyst performs the extraction by applying successively different techniques/tools and it is his/her responsibility to decide that the schema is complete and that all the needed constraints have been extracted.

This shows that DBRE heavily relies on the analyst skills.

4. Process Automation

DBRE principles and methodologies published so far are well understood and can be quite easily applied to small and well structured systems. But when we trying to apply them to real size complex systems, we have to face huge volume of information to manipulate and analyze.

So automation is highly desirable to perform DBRE in large projects within reasonable time and cost limits. It is usually admitted that an analyst can manipulate (manually) 50,000 lines of code [9], but real projects can have ten to hundred times more LOC.

When we speak of automation, this does not mean that the complete DBRE process will be done automatically without the analyst's intervention. Instead, in most processes, the analyst is provided with tools that help him in his work. He has to decide which tool he want to use at a given time and how to interpret the results. Many of the tools are not intended to locate and find implicit construct, but rather contribute to the discovery of these constructs by focusing the analyst's attention on special text or structural patterns. In short, they narrow the search scope. It is up to the analyst to decide if the constraint that he is looking for is present. For example, computing a program slice provides a small set of statements with a *high density of interesting patterns* according to the construct that is searched for (typically foreign keys or field decomposition). This small program segment must then be examined visually to check whether traces of the construct are present or not.

There are several automation levels.

- Some processes can be fully automated. For example, during the schema analysis process, it is possible to have a tool that detects all the possible foreign key that meet some matching rules. Example: the target is an identifier and the candidate foreign keys have the same length and type as their target.
- Other processes can be partially automated with some interaction with the analyst. For example, we can use the dataflow diagram to detect automatically the actual decomposition of a field. The analyst is involved in conflict resolution (e.g., two different decomposition patterns for the same fields).
- We can define tools that generate reports so that the analyst can analyze them to detect the existence of a constraint. For example, we can generate a report with all the fields that contain the key words "id" or "code" and that are in the first position in their record layout. The analyst must decide which fields are candidate identifiers.

5. Limits of the automation

Another reason for which full automation cannot be reached is that each DBRE project is different. The sources of information, the underling DBMS or the coding rules, can all be different and even incompatible. So for each project, we may need some specific tools to discover automatically some very specific constraints in some projects.

As said in the previous sections the data structure extraction process cannot be fully automated because this process basically is decision-based. The discovery of foreign keys into a program source code is an example of constraint elicitation that can not be fully automated. To discover a foreign key using program understanding techniques, we are looking for data dependencies between the fields of two records. But data dependencies do not necessarily materialized a foreign key. It can be a functional dependency between the two fields (as a price and the price with VAT) or it can be some business rules (as the order number is some function of the customer number and the order date). So we can imagine a tools that find data dependencies between database fields, but the analyst need to qualify those dependencies.

But even in activities of the process that can be partially or completely automatize, the tools must be used with some precaution [10]. While tools are likely to provide better results than unaided hand analysis, the analyst needs to understand the strengths and weaknesses of the tools used. There are still many cases in which tools either fail to capture some constraints (missed targets or *silence*) or show constraints that do not really exist (false targets or *noise*). The analyst must validate the results and it is his responsibility to accept or not the constraints proposed by the tools and to decide to look further to find other constraints.

Even if some tools can help the analyst, the tools alone does not automatically lead to increased productivity. Training and well defined methods are needed to get a real benefit. Data structure extraction is a difficult and complex task that requires skilled and trained analysts that know the languages, DBMS, OS used by the legacy system and he must also master the DBRE methodology, techniques and tools (which include database engineering).

6. DBRE project evaluation

DBRE project evaluation can be a critical problem when the DBRE is an explicit process in a project or when it is the project by itself. The customer wants to evaluate the work of the analyst to know if it was well done and if the results can be used as input for another project (as data migration or reengineering).

Unlike classical engineering projects, where the final result is "concrete", i.e., some piece of software that the customer can use and test to check if it meets the initial specification. In DBRE projects, the final result is an abstract specification, made up of logical and conceptual schemas. The customer can find very difficult to evaluate these schemas.

One of the only way to be sure that the DBRE is complete and correct would be to use its results to migrate the application to the new database schema and to check if the new application has the same behavior as the old one. This *a posteriori* approach is only realist when the DBRE is a first step in a migration process and the same team carries out the whole process. We can conclude that reverse engineering projects are more difficult to evaluate than engineering ones.

A realistic approach could be, as suggested above, to agree, at the beginning of the project, on the constraints that are looked for and the techniques and tools used. It is also important to explain to the customer the strengths and the weakness of the chosen approach so that he can evaluate the quality of the result he can be expected.

The critical process with respect to the quality of DBRE is the data structure extraction. Indeed, the quality of the conceptual schema depends mainly on the quality of the logical schema because it is obtained by semantic preserving transformations. The quality of the data structures extracted depends on the analyst skills and tools, on quality of the information sources but also on the time spend during the analysis. Not surprisingly, the quality of the results is thus an economic issue.

7. The Economic Challenge

Despite the power of the supporting tools, the discovery of, say, a foreign key through the analysis of half a million lines of code is not a trivial job. One have to find statements that perform the checking of the constraint, or that rely on this implicit foreign key, and these statements can be spread in all the program and the same constraint can be checked at several locations, possibly with a different algorithm. This analysis takes times and need to be repeated for each constraint. Considering that an actual schema can include several thousands of implicit constructs, DBRE can prove very costly for medium and large projects. We have two solutions to reduce the cost of a DBRE project: to increase the automatic parts of the process or to decrease the completeness, and therefore the quality, of the results.

Let V be some measure of the DBRE project size. The DBRE cost can be expressed as a function of this size: $f(V)$. As already discussed, this function has complexity $O(V^2)$. We can also express the cost as the sum of the manual and the automatic part of the effort:

$$f(V) = f_m(V) + f_a(V)$$

where $f_m(V)$ represents the cost of the manual part and $f_a(V)$ the cost of the automatic part. They are also both polynomial in V, but f_m increases faster that f_a. Those functions depend on the level of automation used and vary from one project to the other.

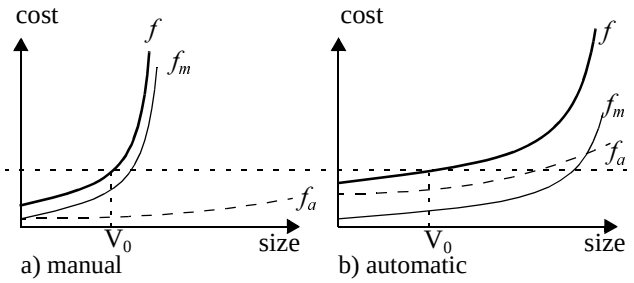

Figure 3. Components of the cost of DBRE projects.

As an example, Figure 3.a sketches those functions for projects of different size carried out mainly manually and the Figure 3.b represents the cost for the same projects, with a highest level of automation as possible.

Those graphs show that for small projects the manual solution is acceptable and can be less expensive than the automated one (if the size is under V_0). But when the size of the project increases, the cost of the manual processing increases very fast. When we use an automated solution, there is an important initial cost (independent of the size) that represents the research of the best tools, the customization or creation of tools. When all this is done, the cost increases very slowly with the size of the project.

As an example, if we took a small application (one program of 1K LOC and 3 record types). An analyst can recover the complete schema of this application manually in half a day. If we took a medium size application (150 programs of 200K LOC and 200 record types). To reverse engineer this application we have adapted our program understanding tools (10 days), it took only three hours to the tools to analyze all the programs and then the analysts need to validate the results (detect 1000 foreign keys and functional dependencies) in 10 days. It is very difficult to do this project manually, because the programs call each other and it is impossible to recover the database schema by analyzing independently each program. To analyze this application the analyst has to follow the inter-program calls. If the analyst takes only one hour to discover each foreign key or functional dependency, it will take about 40 days to complete the projects.

So we can reduce the cost of big project, if we can find or create tools that maximize the assisted part of the process.

The other solution to reduce the cost is to decide to do only partial reverse engineering, thus obtaining an incomplete schema, i.e., we are only looking for the constraints that are the easiest (and cheapest) to find. This partial reverse engineering leads to an incomplete schema with a lot of missing constructs (silence) and perhaps some noise. The drawback of this solution is that it can lead to incorrect results, if it is used as input for a data migration, datawarehouse or reengineering projects. These incorrect results can produce very high cost late in the projects. For example if we try to migrate the data into a new database where the customer and the order are mixed in the same record type and the application is not aware of such characteristics. When the user asks for the customer's list then orders also appear, so he spends a lot of time to find the customer among the order.

The partial reverse engineering can be very useful if we want a first inventory of the records of the database or if the programmer how develop the application is still present and can complete (correct) our schema. It is very important that the analyst and the customer to be aware of weakness of the schema produce by this partial reverse engineering.

So the analyst has to find, with the customer, the right level of completeness of his (her) result with respect to the possible automation, the cost and the precision needed. One of the possible solutions is to proceed step by step. At each step, we decide if we go further and if we continue, we define what we are looking for in the next step. One of the *no-go* criteria, can be the number of constraints discover during the previous step. This is equivalent to stop when discovering new constraints becomes too expensive.

8. Scalability in the DB-MAIN DBRE approach

Several industrial projects have proved that powerful techniques and tools are essential to support DBRE, especially the data structure extraction process, in realistic size projects [9]. These tools must be integrated and their results recorded in a common repository. In addition, the tools need to be easily extendible and customizable to fit the analyst's exact needs.

DB-MAIN is a general-purpose database engineering CASE environment that offers sophisticated reverse engineering toolsets. DB-MAIN is one of the results of an R&D project started in 1993 by the Database Engineering Laboratory of the Computer Science department of the University of Namur (Belgium). Its purpose is to help the analyst in the design, reverse engineering, migration, maintenance and evolution of database applications.

DB-MAIN offers the usual CASE functions, such as database schema creation, management, visualization, validation, transformation, as well as code and report generation. It also includes a programming language (*Voyager2*) that can manipulate the objects of the repository and allows the user to develop its own functions. Further detail can be found in [2] and [4].

DB-MAIN offers several functions that are specific to the data structure extraction process [5]. The *extractors* extract automatically the data structures declared into a source text. Extractors read the declaration part of the source text and create corresponding abstractions in the repository. The *foreign key assistant* is used during the Refinement phase to find the possible foreign keys of a schema.

Program analysis tools include three specific program understanding processors.

- A *pattern matching* engine searches a source text for a definite pattern. Patterns are defined into a powerful pattern description language (PDL), through which hierarchical patterns can be defined.
- DB-MAIN offers a *variable dependency graph* tool. The dependency graph itself is displayed *in context*: the user selects a variable, then all the occurrences of this variable, and of all the variables connected to it in the dependency graph are colored into the source text, both

in the declaration and in the procedural sections. Though a graphical presentation could be thought to be more elegant and more abstract, the experience has taught us that the source code itself gives much *lateral* information, such as comments, layout and surrounding statements.

- The *program slicing* tool computes the program slice with respect to the selected line of the source text and one of the variables, or component thereof, referenced at that line.

One of the lessons we painfully learned is that they are no two similar DBRE projects. Hence the need for easily programmable, extensible and customizable tools. The DB-MAIN (meta-)CASE tool includes sophisticated features to extend its repository and its functions. In particular, it offers a 4GL language through which analysts can develop their own customized functions.

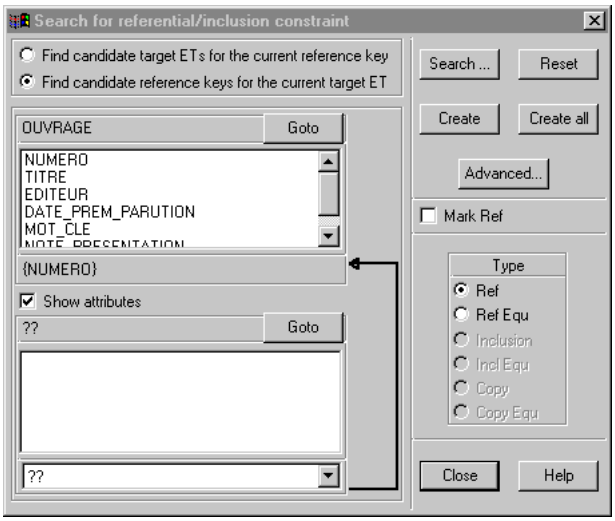

Figure 4. Semi-automatic foreign key assistant. The analyst gives one side (source or target) of the foreign key.

Until 1997, we have developed techniques and tools for *punctual* problems, i.e., that address the recovery of one instance of a construct at a time. Such tools can be called *micro-analytical*. Their usage is tedious in large projects and the time needed to complete the process is dramatically high.

Since 1997, we decided to encapsulate those tools into some mass processing engines. Those engines apply micro-techniques automatically and discover constraints or generate some reports.

We can take as an example the elicitation of candidate foreign keys into a database schema.

Until 1997 (DB-MAIN v3), there were no specific tools to support foreign key elicitation. The analyst had to analyze the whole schema and create the foreign keys manually. This was acceptable for small projects, but it became tedious and error-prone for larger projects.

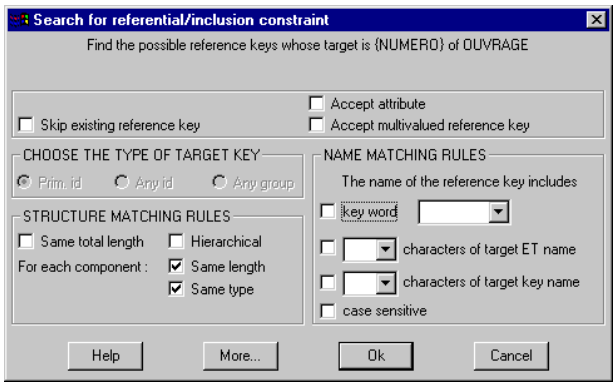

Figure 5. The foreign key assistant matching rules.

In 1998 (DB-MAIN v4), we incorporated a semi-automatic foreign key assistant. The analyst gave one side of the foreign key (typically the target - Figure 4) and the tool found all the possible foreign keys that could reference this target. The analyst had to give the assistant the matching rules to find the other side of the foreign key (see Figure 5). Those rules can be: the two sides of the foreign key must have the same type, same length, or same structure, the origin of the foreign key must contain some keywords or (a part of) the name of the target, there is an index on the origin, etc.

In 1999 (DB-MAIN v5), the foreign key assistant is fully automated. The analyst gives the tool the matching rules as in the semi-automatic approach; i.e., rules that determine if there is a foreign key between two (groups of) fields (Figure 5). The tools needs also a criterion to find all the possible targets of the foreign keys (Figure 6), otherwise the search space will be too large, the number of origin-targets to compare is about $(f! \times r)^2$. Where f is the average number of fields per record type and r the number of record types. $f!$ is the number of possible groups of fields per records, so $f! \times r$ is the total number of groups of fields in the schema. $(f! \times r)^2$ is the total number of possible foreign keys in the schema, if we take any group of fields as the origin and as the target. Usually the target records are the identifiers (the number of origin-targets to compare is only about $(f! \times r) \times r$, because, generally, there is only one identifier per record). The tool finds, for each target, all the possible foreign keys and can generate a report. The analyst reads and modifies the report, which is analyzed by a second tool that creates the foreign keys checked by the analyst.

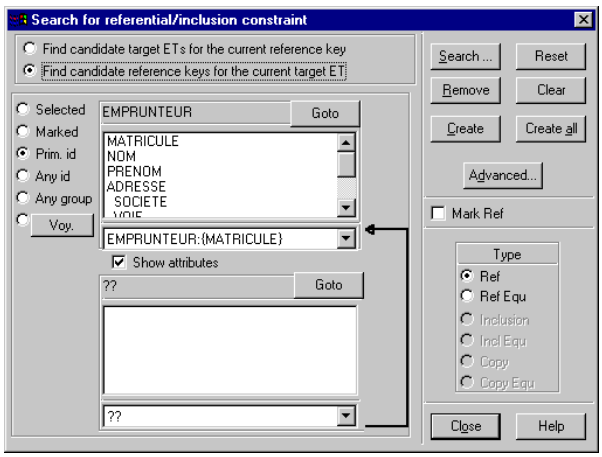

Figure 6. The automatic foreign key assistant. The analyst gives the criteria to find the target of the foreign key.

The analyst needs to give the right matching rules. If too restrictive rules are specified, the tool will generate silence (does not discover some existing foreign keys). If the rules are too loose, the tool will generate noise (suggesting spurious foreign key). Those rules vary from one project to the other. A first (manual) analysis of the project is needed to discover those rules. Sometimes, several sets of matching rules are needed for the same project, for instance when several teams were involved in the design of the database. Another drawback is that the (semi)automatic tool cannot be applied to projects where matching rules are impossible to find or where the matching rules give to much noise or silence. For example, this is the case for databases where there is no meaningful field and record names and where there are many fields (that are not foreign keys) with the same type and length as the identifiers.

On the economic side, the manual approach can be very expensive for big projects, as we saw above. With the semi-automatic approach, the analyst only needs to discover the target of the foreign key and the matching rules (only once per project) and the tool performs the rest of the work (few seconds per target). The cost being the sum of the manual analysis cost (linear w.r.t. project size) and of the automatic analysis cost (also linear, but far less expensive than the manual part). With the automatic approach, the analyst only gives the matching rules (once for per project) and the tool finds all the candidate foreign keys (in a few seconds). Now the cost depends mainly on the automatic analysis cost.

We are trying to increase the automated part of the process to reduce the cost and to allow the reverse engineering of larger projects. One of the problems we are addressing is the validation procedure that ensures that automation does not lead to a loss of quality.

9. Conclusion

In this paper, we have shown the need for tools to automate reverse engineering. Even if complete automation is impossible, some form of it is crucial to perform large project reverse engineering in a reasonable time, at an acceptable price.

On the other hand, the increased use of tools to automate the data structure extraction cannot lead to a loss of quality of the resulting schema. Indeed poor quality recovered schema induces high reengineering cost and unreliable applications.

The role of the analyst is very important. Except in simple projects, he needs to be a skilled person, who masters the DBRE tools, knows their strengths and weakness, is competent in the application domain, in database design methodology, in DBMS's and in programming language (usually old ones).

One of the major objectives of the DB-MAIN project is the methodological and tool support for database reverse engineering processes. We quickly learned that we needed powerful program analysis reasoning and their supporting tools, such as those that have been developed in the program understanding realm. We also learned that tools had to be used with much care and caution. We integrated these reasoning in a highly generic DBRE methodology, while we developed specific analyzers to include in the DB-MAIN CASE tool.

An education version is available at no charge for non-profit institutions (http://www.info.fundp.ac.be/~dbm).

10. References

[1] Batini, C., Ceri, S. and Navathe, S.B.: *Conceptual Database Design - An Entity-Relationship Approach*, Benjamin/Cummings, 1992.

[2] Englebert, V., Henrard J., Hick, J.-M., Roland, D. and Hainaut, J.-L.: "DB-MAIN: un Atelier d'Ingénierie de Bases de Données", Ingénierie des Système d'Information, 4(1), HERMES-AFCET, 1996.

[3] Hainaut, J.-L., Chandelon, M., Tonneau, C. and Joris M.: "Contribution to a Theory of Database Reverse Engineering", in Proc. of WCRE'93, Baltimore, IEEE Computer Society Press, 1993.

[4] Hainaut, J.-L, Roland, D., Hick J-M., Henrard, J. and Englebert, V.: "Database Reverse Engineering: from Requirements to CARE Tools", Journal of Automated Software Engineering, 3(1), 1996.

[5] Henrard, J., Englebert, V., Hick, J-M. , Roland, D. , Hainaut, J-L.: "Program understanding in databases reverse engineering", in Proc. of DEXA'98, Vienna, 1998.

[6] Jerding, D., Rugaber, S.: “Using Visualization for Architectural Localization and Extraction”, in Proc. of WCRE’97, Amsterdam, 1997.

[7] Montes de Oca C., Carver D. L., “A Visual Representation Model for Software Subsystem Decomposition”, in Proc of WCRE’98, Hawai, USA, IEEE Computer Society Press, 1998.

[8] Sneed H.: “Economics of sofw*are re-engineering”. Software Maintenance: Research and Practice*, 3, 1991, pp 163-182.

[9] Tilley S: “A reverse-engineering environment frameword”. Technical report CMU/SEI-98-TR-005, Carnegie Mellon University, http://www.sei.cmu.edu/publications/documents/98.reports/98tr005/98tr005abstract.html, 1998.

[10] Wilde, N.: “Understanding program dependencies”. Technical report CM-26, http://www.sei.cmu.edu/publications/documents/cms/cm.026.html, 1990.